36. (New) A method for an original operating system (a host OS) in a mobile device that supports memory protection mechanism to run another operating system (a guest OS) within the same memory space of said host OS while preserving the current state of said host OS in memory throughout the execution of said guest OS, comprising the steps of:

said mobile device running said host OS;

said host OS starting said guest OS through a launcher;

said launcher go through said memory protection mechanism to mark memory blocks currently used by said host OS as protected from guest OS;

said launcher launching said guest OS;

said guest OS running, accessing only memory blocks that have not been marked as protected so that said memory blocks marked as protected are preserved throughout the execution of said guest OS;

said guest OS finish running through an exit-code;

said exit-code restores the state of said host OS by reverting said protected memory blocks;

said host OS resuming its operation.

37. (New) The method of claim 36, wherein said memory protection mechanism is achieved through a memory management unit (MMU) that allow or disallow a program to access particular memory addresses.

38. (New) The method of claim 36, wherein

said launcher go through said memory protection mechanism to mark memory blocks currently used by said host OS as protected from guest OS further comprising steps of:

moving memory blocks currently used by host OS whose address range is needed by said guest OS to a free memory location in the device;

marking said free memory blocks as protected;

and wherein restores the state of said host OS by reverting said protected memory blocks further comprising steps of:

restoring said free memory blocks to said memory blocks whose address range is needed by said guest OS;

reverting said free memory blocks from protected.

39. (New) The method of claim 36, to start a second guest OS from said guest OS, further comprising steps of:

said guest OS acting as a host OS to the second guest OS;

said second guest OS acting as a guest OS;

Repeating steps in claim 36 to start said second guest OS from said guest OS.

40. (New) A method for an original operating system (a host OS) of a mobile device to start another operating system (a guest OS) while keeping the running state and data of said host OS in memory throughout the execution of said guest OS, comprising the steps of:

said mobile device running said host OS;

said host OS starting said guest OS through a launcher;

said launcher moving memory blocks in lower address space that are currently used by said host OS to free memory blocks in upper address space, and eventually preserving current state and data of said host OS to the upper address space through the following steps:

for each of said memory blocks in lower address currently used by said host OS, find a free memory block in the upper address space; and

move said used memory block in lower address space to said free memory block in upper address space;

said launcher identifying a memory address location where memories above said location contains host OS data and memory addresses below said memory address location to be free to use by said guest OS;

said launcher launching said guest OS, passing memory address location as a reduced memory size to said guest OS;

said guest OS running in said reduce-sized memory space and leave memory space higher than said memory location untouched;

said guest OS finish running through an exit-code;

said exit-code restores the state and data of said host OS by reverting each of said memory blocks in lower address from each of said free memory blocks in upper address;

said host OS resuming operation.

41. (New) The method of claim 40, wherein said launcher launching said guest OS, passing memory address location as a reduced memory size to said guest OS; further comprising:

a memory detection module of said guest OS uses said reduce-sized memory space as the total available memory of the mobile device instead of the real memory size of the mobile device.

42. (New) The method of claim 40, wherein said launcher launching said guest OS, passing memory address location as a reduced memory size to said guest OS; further comprising:

said launcher modifying system register in said mobile device to report said reduced memory size as total available memories to said guest OS.

43. (New) The method of claim 40, wherein memories of said mobile device are divided into multiple memory banks, and said launcher launching said guest OS, passing memory address location as a reduced memory size to said guest OS; further comprising:

Generating said reduced memory size by disabling one or more said memory banks to make them no longer available to said guest OS.

44. (New) The method of claim 40, wherein input output (IO) states, registers of said mobile device within said host OS are preserved and later restored into said free memory block in upper address space.

45. (New) A method for an original operating system (a host OS) of a mobile device to start another operating system (a guest OS) within the same memory space of said host OS while keeping the running state of said host OS throughout the execution of said guest OS in place, comprising the steps of:

said mobile device running said host OS;

said host OS starting said guest OS through a launcher;

said launcher launching said guest OS, passing a list of memory addresses of currently used memory of said host OS to a memory device driver in guest OS during initialization of said guest OS;

said memory device driver of said guest OS claiming said list of memory addresses and keeping them from being modified by any other part of said guest OS during the execution of said guest OS;

said guest OS running in the same memory space of said host OS, with memories used by said host OS being claimed and protected under said memory device driver;

said guest OS finish running through an exit-code;

said exit-code restores the state of said host OS by releasing said list of memory addresses from said device driver;

said host OS resuming operation.

46. (New) The method of claim 45, wherein said list of memory addresses used by host OS include memory addresses only in dynamic runtime memory of said host OS and not memories.

47. (New) The method of claim 45, wherein said list of memory addresses used by host OS also include the current input output (IO) states and current registers of the mobile device.

48. (New) A method for an original operating system (a host OS) in a mobile device to start another operating system (a guest OS) and able to resume to the current state of said host OS after the execution of said guest OS, comprising the steps of:

said mobile device running said host OS;

said host OS starting said guest OS through a launcher;

said launcher saving the image of currently used memories of host OS to an external memory device;

said launcher launching said guest OS;

said guest OS running;

said guest OS finish running through an exit-code;

said exit-code restores said host OS from said external memory device;

said host OS resuming its operation.

49. (New) The method of claim 48, wherein the image of currently used memories of said host OS can be transferred to another mobile device and resumed there.

50. (New) A method of packaging an image of guest OS inside a native application of a host OS to allow in-place execution of the said guest OS image in order to reduce memory usage where the format of said native application contains multiple un-continuous data chunks in memory space, comprising the steps of:

compiling an image of guest OS into multiple code segments;

appending each of said code segments with a jump table containing multiple jump addresses to be used to point to the other code segments;

for each inter-segment jump instructions in each of said code segments, linking each of said jump instructions point to an entry of said jump table;

preparing a native application for said host OS with a startup code and multiple data chunks;

each of said data chunks wrapping each of said code segments of said guest OS plus each of said jump tables;

said host OS launching said guest OS by running said native application;

said startup code in said native application looping through each of said jump table entries to fill in the current memory address of each corresponding code segments;

each of said code segment executing a jump instruction to invoke codes in another code segment;

said executed jump instruction getting real address of said another code segment from said jump tables;

said executed jump instruction successfully jumping to the other code segment wrapped by said another data chunk;

said image of guest OS now executing in place without the need to extract all the code segments from said native application and rearrange them in a continuous memory location.

51. (New) The method of claim 50, wherein said native application in said host OS is a database file with multiple data records or Palm PDB file with multiple Palm DB records.

52. (New) The method of claim 50, wherein said native application in said host OS is a Palm PDB file with multiple Palm DB records and said host OS is Palm OS.

53. (New) The method of claim 50, wherein said image of the guest OS code requires one sequential memory address, said native application of host OS contains multiple data chunks that has a maximum chunk size; and compiling an image of guest OS into multiple code segments further comprising the steps of:

compiling said image of guest OS into one code segment with sequential memory address;

re-arranging instructions in said image of guest OS to reserve spaces to be used for jump tables on each of said chunk-size boundaries by inserting jump instructions to by pass those spaces;

splitting said one code segment into multiple code segments on said chunk size boundaries;

54. (New) The method of claim 50, wherein the size of said guest OS image exceeds the maximum size limit of said native application of said host OS, further comprising the steps of:

splitting said image of guest OS into multiple pieces;

compiling each said piece of said image of guest OS into multiple code segments;

packaging each said piece of said image of guest OS into a native application of said host OS;